DTIC FILE COPY

(4)

ARL-TR-89-32

Copy No. 31

THE APPLICABILITY AND LIMITATIONS OF EXPERT SYSTEM SHELLS

AD-A210 801

James K. Kroger

APPLIED RESEARCH LABORATORIES
THE UNIVERSITY OF TEXAS AT AUSTIN
POST OFFICE BOX 8029, AUSTIN, TEXAS 78713-8029

30 May 1989

Technical Report

Approved for public release;
distribution is unlimited.

DTIC
ELECTE
AUG 4 1989
S A D

THIS WORK WAS SPONSORED UNDER THE
INDEPENDENT RESEARCH AND DEVELOPMENT PROGRAM AT
APPLIED RESEARCH LABORATORIES.

097

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) ARL-TR-89-32	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Applied Research Laboratories	6b. OFFICE SYMBOL (if applicable) ARL:UT
7a. NAME OF MONITORING ORGANIZATION Applied Research Laboratories	
6c. ADDRESS (City, State, and ZIP Code) The University of Texas at Austin P.O. Box 8029 Austin, Texas 78713-8029	7b. ADDRESS (City, State, and ZIP Code) The University of Texas at Austin P.O. Box 8029 Austin, Texas 78713-8029
8a. NAME OF FUNDING/SPONSOR ORGANIZATION	8b. OFFICE SYMBOL (if applicable)
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER Independent Research and Development Program	
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT No.	PROJECT No.	TASK No.	WORK UNIT ACCESSION No

11. TITLE (Include Security Classification)
The Applicability and Limitations of Expert System Shells

12. PERSONAL AUTHOR(S)
Kroger, James K.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
technical	____ TO ____	89-5-30	

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
expert system shells knowledge-based systems, rule-based systems

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

As expert system technology has moved from the laboratory into the field, tools for building expert systems have become commercially available. This report reviews the major aspects of these tools, or shells. It discusses in depth two sample projects using expert systems shells. Finally, it discusses problems which exist with current expert system shells, and makes recommendations for overcoming them.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED [✓ SAME AS RPT.] ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Arnold Tucker	22b. TELEPHONE (Include Area Code) 512-835-3294
22c. OFFICE SYMBOL EMG	

TABLE OF CONTENTS

A-1

1. INTRODUCTION

There is presently a widespread interest in expert system shells. In a survey of tool vendors, the Applied Artificial Intelligence Reporter found that in addition to the seven thousand expert systems already under development, six thousand new systems will be begun this year (Anon, 1987). While it should be noted that these figures are probably more indicative of the number of development tools sold than the number of systems under active development, there is little denying that the number of tool applications is increasing rapidly.

The first successful expert systems, such as the medical diagnostic programs MYCIN (Shortliffe, 1976) and CASNET (Weiss, 1978), demonstrated that a program consisting of a large body of structured information about a domain, along with a strategy for reasoning with this information, could solve domain specific problems as well as an expert in the field. Following their development, work was undertaken to encapsulate the principles which had been used to represent and apply knowledge in programs, or shells, which would be generalizable to other problem domains. The programs which resulted were EMYCIN (van Melle, 1979) and EXPERT (Weiss, 1979). These programs spawned the current proliferation of expert system development tools, which range from integrated software and hardware LISP environments to complete, implemented expert system shells which lack only the domain specific knowledge. These shells are largely based on some combination of the same few knowledge representation principles; differences between them lie mainly in implementation and the way the system is interfaced to the person who will use the tool to build an expert system. Yet there is some uncertainty as to when this methodology is preferable to traditional programming methods, and as to how well these expert system principles as they are embodied in various tools may be applied to diverse, complex real world problems. This report reviews current expert system methodology, describes attempts to fit the expert system shells S.1 and RuleMaster to substantial military applications, discusses the viability of expert system shells for diverse applications, and makes recommendations for overcoming the shortcomings of commercial shells.

2. CURRENT EXPERT SYSTEM METHODS

Expert systems traditionally consist of three parts: the knowledge base, the inference engine, and the user interface. The user interface determines how the builder and user of the system will interact conceptually with the system. Although in code it commonly comprises the largest part of the system (Bobrow, 1986), it is the other two parts that do the deductive work of the system.

The knowledge base consists of a body of knowledge about a given problem domain, coded according to one or more of the dominant schemes for representing knowledge. The primary methods of representing knowledge are rules, frames, semantic nets, objects, and logic. The inference engine is a domain independent strategy for manipulating information in these representations.

2.1 RULE-BASED SYSTEMS

Rule-based systems generally follow the production system format (Nilsson, 1980). In its simplest form, a production system consists of (1) a number of *if-then rules* each consisting of an antecedent (if) and an action (then), (2) a *state*, or set of affirmations about the current status of the problem domain, and (3) a *control strategy*. If condition(s) in the antecedent of a production are true with respect to the state, then the action may be taken. The action causes changes in the state. Then, the next rule with a true antecedent is fired, and so on, until a desired state is achieved. This is known as *forward chaining*: inference begins with an antecedent and ends with an action. A production system can also be *backward chaining*: a conclusion is posited, and a rule with a matching conclusion is found. The truth of the antecedent of that rule is tested. If it is true, the truth of the posited conclusion is affirmed. Determining the truth of the antecedent may require treating it (or part of it) as another conclusion and searching for a matching rule again, thus chaining backwards from conclusions to antecedents until a true antecedent is encountered.

In systems with a vast array of rules, it may be inefficient to simply search for the next rule with a true antecedent; moreover, when the antecedents of more than one rule are true, there must be some method to choose among them. These issues are addressed in the control strategy. The latter issue is called conflict resolution (Michaelson, 1985). Example search strategies are to search only a subset of the rules for one to fire, and for each fired rule to specify the rules to fire next. A common conflict resolution strategy is to choose the rule that most (or least) recently became true.

The domain-specific rules, or those which pertain to the problem domain, are combined with declarations about the problem domain, or rules without an "if" side, to form the knowledge base. They describe a particular domain and are not transferable to other domains. The control strategy will include the means for forward chaining or backward chaining (or both). It may also contain meta-rules to guide rule selection and conflict resolution. In theory, the control strategy is the domain-independent knowledge, an abstract method of rule-based deduction, which can be applied to different knowledge bases for different domains. This control mechanism is the inference engine in rule-based expert system shells. The inferencing to be modeled in large expert systems can be quite convoluted, necessitating the addition of heuristic meta-rules or special control functions, with the effect that some domain knowledge is implicit in the inference engine. The resulting control strategy is often not generalizable to other applications. That is why, as we shall see, the inference engines in rule-based expert system shells tend to involve fairly simple strategies.

2.2 FRAME-BASED SYSTEMS

A *frame* is the unit of information in the frame knowledge representation scheme. A frame represents an object with a list of the object's attributes (Minsky, 1975). A primary feature of frame systems is *inheritance*. An example will help demonstrate how inheritance works: suppose a frame to represent the object "car", with one of its attributes being "wheels". The value of this attribute is four. Now suppose another frame for the object "Chevrolet". One of the attributes of "Chevrolet" is that it is an instance of the class "car". Thus, by default, the frame "Chevrolet" inherits for its attribute "wheels" the value 4. In like manner, if the value for one of the attributes of "car" had not been specified, it would be possible to check the value of the attribute for "Chevrolet" or any other instance of car for an example value (in some inheritance schemes, only the attributes are inherited, not the values of the attributes (Mettrey, 1987)). Inheritance allows the identity and class of objects, as well as the values of their attributes, to be inferred (Rich, 1983).

Abduction is another strategy for classification with frames (Ramsey, 1986). Hypotheses, or frames from the knowledge base, are proposed to account for the presence of the attributes (or values of the attributes) of a particular object. Then the ability of each hypothesis to account for the attributes of the object is tested by searching for a match (or connection) between the attributes of the hypothesized frame and the object. This may produce a classification, or more hypotheses, with the intended result that a sufficient match is eventually found.

Semantic nets are similar to frame systems, the difference being that connections between objects in semantic nets can be of any kind, rather than merely instantiation. An example is PROTOS (Porter, 1986), a learning diagnostic expert system which represents types of maladies with category frames and particular cases with exemplar frames. Attributes, or features, of the categories and exemplars are also represented by frames, and may themselves constitute categories. There are several kinds of connections between exemplars, categories, and features, such as "causes" and "is a function of". Protos reasons abductively. A new case is represented by a frame. If a suitable chain of connections between the features of the new case and those of a category or exemplar can be found, the category (or the category to which the exemplar belongs) is determined to be the diagnosis.

The inference engine of frame-based expert system shells employs techniques such as these. Also included with the system are templates for creating frames. The domain dependent part of a frame-based system is the collection of frames representing the objects of interest in the domain and the connections between them.

2.3 OBJECT-ORIENTED SYSTEMS

The object-oriented approach is also similar to the frame representation scheme (Klahr, 1980). Objects are represented by frame-like structures, and are organized hierarchically. In addition to attributes, these frames may also contain methods (procedures), which may be inherited. Objects are able to send messages to each other. When an object receives a message, one of the object's methods is activated. Object-oriented programming is not limited to expert systems, but it has been employed as a knowledge representation technique to portray complex interactions between entities in a domain. It is a feature of some shells, such as Art and KEE (Mettrey, 1987).

2.4 LOGIC PROGRAMMING

Logicians have long rendered propositions about the world in a formal language. They are able to use logical rules of inference to prove these propositions. However, the variability of these rules made their use in a computer program impractical. With the introduction of resolution (Robinson, 1965), a single, powerful inference method became available which was viable for implementation. This method is used by the Prolog programming language interpreter. Knowledge in Prolog is represented by logical predicates and conditionals, which resemble if-then rules in form. Prolog is in effect a backwards chaining inference engine (Cohen, 1985).

2.5 IMPLEMENTATION LEVEL

Another way of classifying expert system development is described by Bobrow et al. (Bobrow, 1986). "The low road approach" refers to direct programming, usually in Lisp or Prolog. It employs an integrated AI programming environment, where the program may be easily developed as it is conceived. The high road approach incorporates an extensive explicitly represented knowledge about the subject matter which can be used for more than one purpose. This method is often referred to as "deep reasoning" (Hayes-Roth, 1984); it involves reasoning from knowledge of the underlying principles and characteristics of the subject matter using causal models, categories, abstractions, and analogies (Michaelson, 1985). Consequently, long inference chains are required to achieve practical results. Deep representations typically employ large networks of frames. This type of system is sometimes too slow and incremental for practical use.

Middle road systems lie between the two above approaches. They also involve explicit representation of knowledge about the problem domain, but are more concerned with practicality and efficiency than with completeness, and use canned representations and procedures which support heuristic problem solving. This approach uses surface representation -- rules which capture only the amount of information about the subject matter needed for practical purposes. Most expert systems developed using shells fall into this category, though it is possible to use the knowledge representation capabilities of some shells to build large, deep representations.

3. CHOOSING A SHELL

Which approach is right for a given problem? It is usually not the case that only one approach will work, though one may be a better choice than others. Niwa, Sasaki, and Ihara (Niwa, 1984) report a comparison of four pilot expert systems to solve the problem of risk management for large construction projects. The systems were a simple rule-based deduction system, a structured rule-based system (rules were grouped according to temporal order), a frame-based system (employing inheritance), and a logic-based system. While more difficult to implement, both the frame-based and structured rule-based systems offered significantly better efficiency than the other two. Comparing a rule-based and a frame-based system for aiding in software engineering management tasks, Ramsey (Ramsey, 1986) found that the two systems performed moderately well, but concluded that the frame-based system was less appropriate since the software engineering field is ill-defined and involves many uncertain relationships.

Rule-based and frame-based systems have their different strengths and weaknesses. Rule-based systems tend to be better for quick-and-dirty solutions to smaller problems. It is usually easier to add or subtract rules from a knowledge base than frames (with their connections to other frames in the knowledge base). Rules are a good choice when the knowledge base lacks structure. Rules are also indicated when the knowledge already exists as rules or in a table format; the type of classification is primarily categorical.

Frame-based systems are especially apt at representing knowledge about numerous entities with well understood, complex relationships. Any application which will need to reason in flexible ways using deep reasoning about the domain is best implemented with a frame-based representation. Frames are also appropriate for knowledge that exists in a descriptive format such as in a textbook or as a single interconnected process, or for problems with multiple simultaneous outcomes (such as competing diagnoses) (Ramsey, 1986).

In deciding if there is a shell which is suitable for a particular problem, one should note that some shells attempt to incorporate more than one approach for representing and inferring from knowledge (Takenouchi, 1987; Fikes, 1985). This is especially true of the larger, substantially more expensive packages, such as Art, S.1, and KEE (Mettery, 1987).

However, it may be difficult to justify the high cost of these systems if they are still insufficiently flexible to handle a large complex application or a variety of different applications. This point will be addressed at greater length in a following section.

3.1 IS AN EXPERT SYSTEM NECESSARY?

Before deciding if any given expert system shell is adequate for our needs, it would be wise to ask an all-to-often unasked question: is an expert system really needed?

"High-level" expert performance can be produced without an expert system as in the case of an auto-pilot (Shore, 1985). One may wonder whether a compiler or a payroll program constitutes an expert system. Knowledge exists in both, of a language and a computer in the former and of accounting and taxes in the latter. These systems could be built as expert systems. The choice of implementation in this case depends on their function: when built using conventional algorithmic techniques, the programs perform a specific, non-flexible task very efficiently. If they were implemented as expert systems, they would include bodies of knowledge which could be used for different purposes, such as answering different questions about the subject matter (Bobrow, 1986).

4. CASE STUDY: S.1

In order to better appreciate which kinds of problem demand an expert system approach and which kind are better handled with traditional techniques, as well as how a shell might expedite expert system development, researchers at Applied Research Laboratories attempted to construct an expert system to solve a complex vehicle pathing problem using the development tool S.1.

4.1 THE S.1 TOOL

S.1 is an expert system shell which supports the use of frames, with inheritance and rules. It uses a backward chaining control strategy for inferencing with rules. The inheritance feature allows inheritance of frame attributes only; it does not allow inheritance of the attributes' values. Each attribute may include a certainty factor. S.1 also supports a Pascal-like language, which can be used to create objects and assign values to their attributes, or to initiate backward chaining. Single integer or character values can be passed to external C functions, as well as a set containing the attribute values and certainty factors of a particular frame. This set cannot be passed into the shell, however. S.1 is implemented in C (Teknowledge, 1985).

4.2 THE PROBLEM

The pathing problem consists of determining a best path to a designated target for a ship-fired missile. The missile receives its guidance instructions from the operator on board the ship before launch. The missile then makes prescribed changes in its direction at preset points, called "waypoints". Path rating criteria are the length of the path and the amount of "threat" the missile is exposed to. The missile must avoid hostile and friendly ships en route to the destination. The precise location of some ships is not known; rather, they are assigned an area of uncertainty (AOU). Threat is accumulated as the missile travels through one of these areas, depending on characteristics of the ship and the AOU. Waypoints are selected at or near the limit of the ship's AOUs, and ship's move in time. Thus, each waypoint would only be determined after the previous one (and its concomitant time of arrival) had been selected. Only 'hen could the locations of ships be predicted in the selection of the next waypoints.

Selecting paths for multiple missiles launched from different platforms with a common target and arrival time in a crowded field of view is a highly complex task, and difficult (if not impossible) for shipboard operators to do. It was felt that a knowledge-based system might have the capacity to accommodate the complex and constantly changing conditions, choose satisfactory waypoints, and thus arrive at the optimal paths. In this sense the system would be responding intelligently to a situation, so the problem seemed a good candidate for an expert system. To begin, the problem was reduced to finding a path for a single missile.

However, because no decisions were required which depended on uncertain or ambiguous knowledge, and no large body of knowledge had to be consulted at any single step, it was found that a combination of a heuristic search algorithm (a variant of the A* algorithm), combined with traditional number crunching, provided an optimal pathing solution. The key point to be made here is that there was always, at each stage in the process, one clearly correct answer: the path-so-far combined with the best next path segment, and it was determined using mathematics and an invariant algorithm. It was possible to code this procedure in a traditional procedural manner. At each stage, the mathematical tests to apply were immediately available as successive components of the algorithm. No searching for the next decision rule was involved.

The frame feature of S.1 proved to be a convenient way to represent the missile and the various kinds of ships; the inheritance feature made it possible to designate, for example, a specific battleship, which would then inherit all of a battleship's features. Inheritance was not used in any inferential way, however. Nor were any rule-based inference methods used, since no rules were implemented.

S.1 features the ability to interface to external C functions. After an initial implementation incorporating some rules for pathing failed inexplicably to execute properly, the pathing functions were written in external functions. Execution of the S.1 shell was initiated; it then called the external functions to determine the best path for a given scenario. A problem was encountered in attempting to reference the same complex data object inside and outside the system (i.e., ships). Internally to S.1 the objects were implemented as frames; externally they were dealt with as C structures. It was necessary to disassemble each object and pass the simple data it was composed of through the interface, and then reconstruct the object on the other side of the interface. The severity of these problems was not revealed until implementation of the system was underway, possibly because it was not anticipated that the majority of the algorithm would exist outside the shell. This underlines a potential difficulty in using a pre-constructed expert system shell for which the source code is not available (a *closed* shell): if the system is unable to perform a needed task, or if the system has a bug, it is not possible to make corrective changes to the program.

After it became obvious that an expert system was not the best solution to the problem, plans to do pathing for multiple missiles were abandoned. However, the experience did suggest some conditions for which an expert system would be more appropriate: if the number of criteria for the determination of best path was greatly increased, efficiency considerations might dictate an "expert", heuristic handling of this knowledge. And if a large amount of symbolic knowledge concerning high-level tactics or strategy were involved, which could not be reduced to mathematical operations, an expert system would be indicated.

In Building Expert Systems, Stefik et al. (1983) list the kinds of tasks performed by experts which are suitable for implementation as expert systems: interpretation, diagnosis, monitoring, prediction, planning, and design. Characteristics common to all of these tasks are large solution spaces, tentative reasoning, time-varying data, and noisy data. Michaelson et al. note that experts "typically solve problems that are unstructured and ill-defined, usually in a setting that involves diagnosis or planning. They cope with this lack of structure by employing heuristics, which are the rules of thumb which people use when a lack of time or understanding prevents an analysis of all the parameters involved." (Michaelson, 1983, p. 303). Expert systems are best suited to problems for which a correct solution is not immediately available, where no clear solution is easily available, and a large body of knowledge must be searched heuristically for a reasonably good answer. Although for a human operator the missile pathing problem appears as a large, unstructured problem, when the problem is subjected to analyses, a clear, very highly structured, and unambiguous solution of the problem was discovered. Aside from the heuristic function employed by the A* algorithm (a heuristic estimate of the quality of a segment of the missile's path), no heuristics were needed. It did not appear that if the problem were expanded to pathing for multiple missiles, the balancing of data about the multiple paths would have been sufficiently difficult to require an expert system.

The most important point here is that an expert system is generally (most) appropriate when search, especially heuristic search, of a large knowledge base is required. This may be due to the fact that the domain knowledge involved lacks structure or definition. However, in other cases, the knowledge may be precise and well structured, but in such large quantity that searching it exhaustively for the single best answer is too inefficient to be practical.

Another indicator that an expert system is needed is the symbolic nature of the knowledge base (Brachman, 1984). This will usually be the case when the knowledge is acquired from an expert. Expert systems are a member of the class of AI programs called "physical symbol systems" (Newell, 1962). They allow the knowledge of the expert to be represented symbolically. In this regard the symbol-manipulating character of Lisp contributed to the conception of expert systems.

5. CASE STUDY: RULEMASTER

5.1 RULEMASTER

RuleMaster, a product of Radian Corporation, is a rule-based expert system shell written in C. Rules are entered in templates called *induction modules*. These modules consist of a *conditions* section, where the parameters of the rule's antecedent are entered along with their possible values; an *actions* section, for listing the actions to be taken as a result of various rule outcomes; and an *examples* section, in which the various combinations of values for the parameters are paired with the appropriate actions. With each parameter, or condition, the method for determining the value of the parameter is specified (such as asking the user for a value or designating another rule module to execute which will return a value for the parameter). The actions taken might include advising the user of a solution, or the execution of other modules. One module serves as the root module, and from each module a tree of modules may extend down from any of its antecedent parameters (for use in determining the value of the parameter) or from any of the actions in its action section. Variables may be passed between calling and called modules. Execution halts when any rule advises the user of a solution.

The rules in the modules are compiled into Radial, a Pascal-like language, which may be then compiled into C. The system builder may program directly in Radial (in a Radial module) when a mathematical or other non-rule procedure is needed. Radial supports integer, floating point, and string data types, as well as arrays. A record-like data structure may be put in a special storage module, which is then accessed like a Pascal record. In addition, RuleMaster can interface to external C data structures and functions through special modules.

RuleMaster uses a very simple inferencing strategy: each rule designates which other rules are to be tested if it "fires". These are the other rule modules specified in the action section. Also, the determination of a value for a condition may require the execution of a designated rule module. This constitutes a simple forward and backward chaining strategy in which no search for rules is required. As a simple example, a rule for determining the weather forecast might have the parameters humidity, temperature, and sky. The humidity parameter might be integer values, or symbolic values like "rising" or "dropping". The temperature parameter could also be of either variety. The sky parameter might have values like "cloudy", "clear", etc. The value of these parameters might be determined by querying the user (the possible values to choose from are supplied), specifying another rule to execute, or specifying a Radial module to execute. For example, the value of humidity might be determined to be "rising" by executing a Radial module which queries the user for the current humidity and compares it to a previous value, then returns the value "rising". The rule module's examples section would contain all the possible combinations of values for these parameters. For the combination "humidity: rising, temperature: 47, sky: cloudy", the corresponding action might be to advise the user "it will rain," or to execute a rule module which will determine some response to be taken or will further analyze the weather situation.

The user interface, consisting of the rule-inducer, the template used for creating rules, Inded (the rule module editor), Sysed (an editor for manipulating the hierarchical structure of modules), module testing functions, and compilation features, comprises the bulk of RuleMaster. This interface proved to be an effective way to enter rules when a small auto fuel system diagnostic tool was created. The values of all of the parameters in that system were determined by querying the user.

5.2 THE PROBLEM

As a further investigation of expert system shell utility, the shell RuleMaster was applied to the problem of casualty assessment in battle simulations. The intention was to replace the casualty assessment portion of an existing system, the automatic data collection system (ADCS), with an expert system built using the shell. One of the perceived benefits of implementing ADCS as an expert system was the ability to easily change the method of casualty assessment by adding or subtracting rules. Also, RuleMaster features the ability to induce rules from examples, and then produce the rules as C code. It was hoped that this might make acquisition and implementation of casualty assessment rules easier. Finally, a characteristic of the ADCS program is constant monitoring of all the participants in the battle simulation. A valuable addition to the system would be the capability to replace the adversary force with an intelligent scenario generator, which would have the ability to respond dynamically to actions of the force in training. Even if the simulated scenario were limited to small portions of the total battle simulation, the savings in manpower would be substantial. Additionally, different responses to the same battle scenario could be compared. The preliminary design called for the use of rules to embody standard tactical responses to the actions of the training force in the ongoing simulation.

The ADCS node to be replaced, the realtime casualty assessment (RT) node, performs two primary functions: keeping track of the position and status of each participant, and determining the outcome of engagements. It can keep track of up to 200 participants, which are equipped with lasers to simulate weapon firing and sensors to detect a hit. The ADCS system polls all of the participants in each 3 second interval. When a participant returns a message that it has fired or that it has been hit (with the identification number of the platform that hit it), this information is stored for a certain number of cycles, and an attempt is made to match fire and hit messages. When a match is made, an analysis of the encounter is made to determine the result of the encounter, which is broadcast back to the involved platforms in the next polling cycle. Three tables provide the probability of kill for various combinations of weapons and platforms at various distances.

When an attempt was made to apply the shell to the ADCS problem, several incompatibilities were discovered. In the ADCS system several enumerated types were used for such things as vehicle type. Enumerated types were not available in the Radial language. In addition, the numoer of array dimensions that could be handled was ambiguous. Although a multi-dimensional array would have been a good way to combine the various tables used to assess engagement outcome, each two-by-two table was eventually entered as a record module with several single dimension arrays. No facility is provided with RuleMaster for loading external data into internal representations; it would have been necessary to build such a mechanism in an external C function, and pass each value into an interface module, which would be used by a Radial module to put the value into the data structure. A list data type was used in the ADCS system, but although RuleMaster provides a simple list-handling capability, it was unable to manipulate the list adequately. Another problem encountered is similar to S.1's inability to pass complex data types through the interface: the modules through which RuleMaster interfaced to external C functions could only pass integers, strings, and floating point values. Thus the elements of arrays, as well as the fields in record modules, would need to be extracted from their data structures before they could be passed to or accessed by external C functions. Similarly, no complex data structure existing externally could be referenced from within any RuleMaster modules.

One of the data structures used heavily in the ADCS RT node was a linked list of records. Since RuleMaster has no record pointer facility which will support linked lists, it was not possible to represent this data structure in RuleMaster. Instead, an array of records was attempted. Each record represented an encounter in which a particular vehicle had been engaged, and included integer and string fields. The array of these records represented all of the unresolved encounters this vehicle had been involved in. This array was a field within another record, which represented all pertinent information about a vehicle, including the array of encounters, as well as its type, the type of weapons it carried, its ID number, and so on. In the original implementation, this larger record contained fields of type integer, string, list, and array, and enumerated types to represent all of this information. This larger record was itself an element of the array of all two hundred of the weapon platforms. The construction of or limitations on complex data structures like this was not discussed in the RuleMaster manual. After an implementation was attempted (the syntax of which was derived largely through guess work since the manual did not mention it) it was confirmed that RuleMaster does not support data types with this degree of complexity. It would allow a field of a record module to be an array of simple integer or string elements, and it would support an array of records, but none of the fields of the records could be an array. Only after the package had been acquired was it discovered that the manual did not address many of these issues and limitations. It was only through the excellent user support provided by Radian that these problems were discovered to be results of the system's design, rather than syntactical or other errors on the part of the user. Were it not for this support, a considerable amount of time might have been wasted in trying to adapt an inappropriate solution to the problem.

As it turned out, re-implementation of the RT node using RuleMaster would have largely consisted of functions and structures external to RuleMaster. Much of this external code and the code internal to RuleMaster would have served the purpose of overcoming the inadequacies of RuleMaster for the problem at hand, such as decomposing complex abstract data types to be passed into RuleMaster, or the extraction of data from tables to be passed as simple values into RuleMaster. This amount of compensation for the tool seemed inordinate to the benefits to be gained -- a very simple (and unalterable) inferencing strategy, rule induction, and the user interface. Thus it was decided that RuleMaster was not suitable for the ADCS application.

6. LIMITATIONS OF EXPERT SYSTEM SHELLS

6.1 CONSIDERATIONS FOR ACQUISITION OF A SHELL

Implicit in the fact that many vendors supply expert system tools without the source code is the belief that prefabricated expert system designs will be applicable to a variety of problems. Doubtless there are many applications for which the shells discussed above are a good "fit". However, as has been demonstrated, this is not always the case. Since expert systems are relatively new, the degree of sophistication necessary to judge the quality of fit between a shell and a problem is not yet widespread. Unfortunately, many purchasers of shells are likely to discover a mismatch through experience. In each of these instances, if the shell is a closed system (the source code is not provided), it represents a wasted investment.

The appeal of expert system shells, with their often touted "instant usability", is especially strong for organizations in which there is a desire to infuse expert system technology into their operations while avoiding both the cost of hiring experienced knowledge engineers and the delay inherent in developing the requisite AI skills internally. It is for just these parties that choosing an expert system shell can be a risky proposition. Our experience indicates that making this choice without an understanding of expert system principles born of experience can amount to a (perhaps unavoidable) shot in the dark. The notion that an expert system shell will allow one to build an expert system without a solid grounding in the principles of their operation can lead to disillusionment. While uncomplicated applications will exist which are fortuitously well handled by the chosen shell, it is possible that the purchaser will find himself with a solution which he cannot apply to his problem. If the problem is very involved, it is likely that the person building the system will end up devoting substantial effort to learning expert system concepts before a working expert system is produced. In comparing the learning that will be necessary for persons of differing experience, it becomes apparent that the abovementioned organizations should (not surprisingly) be prepared to either hire an experienced professional or devote the time necessary to develop the skills through study and experimentation. As is demonstrated in the above examples, applying an expert system shell can require an understanding of traditional data structures and algorithms. Assessing which knowledge representation scheme or inference strategy is appropriate for a task requires an understanding of expert system methods. Also, each shell has its own characteristics and features which will need to be learned, such as the rule templates and Radial language in RuleMaster, and the Pascal-like language, frame format, interfacing procedures, and particular inheritance scheme of S.1. The less experience one has with expert system principles, the more difficult this will be.

A seasoned programmer has a wealth of knowledge about data structures and algorithms which he uses to design solutions to problems. This collection of programn.ıng tools has been augmented with the advent of expert system technology. However, in an attempt to make this new technology available to the uninitiated, as well as to make its application quicker and simpler by providing preconstructed algorithms, expert system shell vendors have created tools which by their prepackaged nature restrict not only the expert system technology which may be brought to bear (witness the simple inference strategies available with most shells (NASA; 1987, Teknowledge, 1985)), but also the traditional programming methods which may be employed. In straightforward problems such as the auto fuel system diagnosis program described earlier, which merely asked the user to choose among a few options at each stage, this might not present a problem. However, few programmers facing a complex, multi-system problem would attempt to apply a data structure or algorithm written by someone who had never seen the problem. The larger, more expensive shells like Art and KEE offer more inferencing and knowledge representation methods than RuleMaster, but should the purchaser feel secure with the far greater expenditure? Will he encounter the same restrictions on his ability to properly address a problem? This depends on the problems he will be attempting to solve. Only if the purchaser knows in advance what problems the shell will be used for, and is able to evaluate the fit of each problem to the shell, can he be assured of having his needs met.

Recently, Frederick Hayes-Roth, the founder of Teknowledge, was interviewed in *Expert Systems*. Teknowledge is the company from which S.1 is available; it also delivers finished expert systems. The likelihood of encountering problems for which even the larger expert system tools such as S.1 are inadequate may be better appreciated by examining some excerpts from the interview:

> *ES*: Have you established an in-house methodology yet?
>
> *FH-R*: Yes, for applications which are structured selection-type applications that can be done very effectively with our own commercial tools...We subscribe to the theory that it's a new field but it does work. There are hundreds of things to apply it to and many of them are very valuable. Therefore it's easy to find good uses for it with moderate to high return and low risk.

ES: Do you find people come to you with projects they want to do that do not fit the kind of return and risk ratios you like to see?

FH-R: For a significant percentage -- probably more than 10 and less than 50% -- of all the problems people present to us, we recommend that they defer it or that they consider it as a research problem as opposed to a development problem. (Anon, 1987).

The literature offers other evidence that problems will be encountered for which a shell may not be appropriate. The symbolic-type processing of the shell often requires augmentation with procedural and numeric computation. Kitzmiller and Kowalik discuss the merging of numeric and symbolic processes in "coupled" systems:

Often, there is no choice. Separately, neither symbolic nor numeric computing can successfully address all problems in design and analysis. Complex problems such as the control of a fully autonomous robot or the design of an aircraft cannot be solved by purely symbolic or numeric techniques. In such cases, a mix of numeric and symbolic techniques is needed to obtain a solution. (Kitzmiller, 1987)

It has already been shown that effecting a smooth marriage between large number crunching processes and the symbolic inferencing processes of both S.1 and RuleMaster can be difficult, if not counterproductive.

Further examples of system needs that might not be adequately addressed by an expert system shell are discussed below.

6.2 FLEXIBILITY IN PURSUING DIFFERENT APPROACHES IN AN INTEGRATED SYSTEM

Examples of problems that may be solved using more than one representation scheme have already been discussed. For some large applications, the scheme that is most appropriate might vary depending on the desired capabilities of the system. Gilmore reports the separate development of target recognition systems by Hughes Aircraft and Martin Marietta Aerospace (Gilmore, 1985). Hughes produced an object-oriented system which incorporated a spatial black board, and groups of model objects in a semantic frame hierarchy. Target recognition was accomplished through abductive classification of objects, using a manner similar to the one used by Protos. Martin Marietta chose to integrate the intelligent part of the system with additional algorithms for global region classification, motion target identification, passive ranging, and advanced object recognition. Along with ancillary data such as digitized maps, time of day, and weather conditions, these algorithms produce input for the expert system portion: a rule based system which assesses the positive and negative contextual evidence for identification. This system provides extensive processing before the expert system is invoked which is not done in the Hughes system. Were an attempt made to solve these problems using shells, considerably more external programming would be required for the latter approach. In the latter case, interfacing a closed shell effectively to all of the various additional algorithms would have been difficult if not impossible. The shells would have different knowledge representation requirements as well. More importantly, had the systems been developed sequentially, using closed shells, by the same organization, it is entirely possible that the shell chosen for the first project would not be suitable for the second one, a fact which might become apparent only after the shell used for the first one were applied to the second. The knowledge representation and interface characteristics of the shell can impose severe limitations on how much a system can be enhanced or altered to meet new requirements.

6.3 ECONOMICAL SOLUTION OF MULTIPLE PROBLEMS

Here some monetary consequences of the preceding problem are emphasized. Cupello, with Morton Thiokal, and Mischelvich, a consultant with Intellicorp, relate their experiences with developing an expert system using a large commercial shell (Cupello, 1988). The total project cost was $206,000. Of that total, $40,000 was paid for the shell (KEE, a Lisp based shell), $65,000 was devoted to the purchase of two Lisp machines, and another $70,000 went toward training, consultation, and education. A major share of the latter expense was devoted to learning the expert system tool used. The focus of the article was the managing of a first expert system project in a large corporation for the purpose of acquiring expert system technology. No description of the shell, the application, or the finished system is given.

The authors completely fail to address the fact that over $100,000 -- more than 50% of the project cost -- was committed to learning and incorporating expert system technology in such a way that actually limited the company's ability to apply it. It committed the company to a choice of attempting to apply the same expert system approach (shell, hardware, training, and consultant) to all subsequent expert system problems or incurring a major expense in acquiring or developing a different approach. The implicit assumption seems to be that the shell invested in will be adequate for all problems; this is highly questionable. In fact, Art and S.1, two of the larger shells, have been recently implemented in C in response to a demand for faster-executing expert systems. Further, it is assumed that the system developed was not intended for wide-spread distribution or marketing; otherwise the authors would certainly have addressed the portability problems inherent in their approach: each system would require its own Lisp environment, probably a Lisp machine. These considerations, along with previous evidence presented, make a large financial commitment to a closed shell for a single application appear ludicrous, especially in light of the small percentage of "finished" expert systems that actually find their way into daily usage (Van De Reit, 1987). In fact, Cupello and Mischelvich do not mention whether their system is in use.

6.4 THE ABILITY TO INTERFACE TO DATABASE MANAGEMENT SYSTEMS

Recently, the integration of expert systems with database management systems has become a topic of interest (Al-Zobaide, 1987). There is a basic structural incompatibility between the two, since database management systems (DBMS) stress syntax over semantics (structure over meaning) while expert systems do just the opposite (Schar, 1988). Any application which depends on data residing on online secondary storage, rather than just user- or sensor-supplied data, is a candidate for an integrated expert system and DBMS. However, the number of expert system shells with the capability to interface to a DBMS is small. These can only interface to one or a few specific DBMSs. In the event that this emerging methodology is deemed most appropriate for an organization's long-term needs, and a decision is made to purchase one of these systems, the limitations of the requisite DBMS must be endured as well as the limitations of the expert system shell.

6.5 THE ABILITY TO WORK IN REALTIME

There is a trend towards applying expert system technology to dynamic processes. In a review of realtime knowledge based systems, Laffey et al. (Laffey, 1988) note that "The increasing complexity [of realtime computer systems] has caused considerable interest in the use of knowledge-based techniques for realtime applications." Among the candidate applications they cite are satellite control, the Pilot's associate, battle management, communications network monitoring and control, and process control. These problems demand an array of special capabilities. Several system requirements are mentioned, including nonmonoticity (changing data input), interface to external environment (gathering data from sensors rather than querying the operator), temporal reasoning, and integration with procedural components (which will perform such tasks as data compression, signal processing, and feature extraction). Requirements such as these compound the problem of choosing a shell. It is not surprising that of 38 systems Laffey described, only ten were developed with the aid of expert system development tools, of which one was modified and one was declared inadequate.

6.6 THE ABILITY TO HANDLE COMPLEX MILITARY APPLICATIONS

Traditional expert system inferencing techniques are unable to cope with the complexity of military scenarios with sufficient speed, even on high-speed processors. Researchers are developing shells with new inference strategies and search techniques in order to avoid the unacceptable limitations on knowledge base size and to better incorporate the problem solving paradigms underlying military decision making (Sorrells, 1985).

7. DISCUSSION

What advice can be drawn from these observations? There are several things to consider before committing oneself to any closed expert system shell. First, expert system methodology must be well understood. Only then can an analysis of a problem be done to determine whether it is appropriate for an expert system, and what methods are required for its solution. Even if it is then determined that a given shell has the capabilities needed, caution is advised; remember the specification uncertainty principle:

> System requirements cannot ever be stated fully in advance, not even in principle, because the user doesn't know them in advance - not even in principle. To assert otherwise is to ignore the fact that the development process itself changes the user's perceptions of what is possible, increases his or her insights into the application environment, and indeed often changes that environment itself. We suggest an analogy with the Heisenberg Uncertainty Principle: any system development activity inevitably changes the environment out of which the need for the system arose (McGracken, 1982, quoted in Kornell, 1987).

Design changes may become desirable during development, but a closed shell may not permit them, since it cannot be altered.

Second, even if it appears that the shell is a good choice for a problem, the requirements of future applications should be taken into account, unless one is willing to absorb the expense in "retooling" for each new application. If the shell under consideration is one of the larger tools, such as Art or KEE, chances are better that it will be suited to subsequent applications. However, the expense (including consulting fees, time spent learning the system, etc.) is much greater, and the system might still be inadequate for some applications. This is most likely if future problems will involve realtime performance requirements, military applications, integration with elaborate external procedural components, or very convoluted inference strategies.

Much of the time saved by purchasing a "complete" shell is realizable only when additional money is spent to hire an expert systems professional or for the services of a consultant. The primary advantages gained with either of these options are an understanding of what problems are best solved with expert systems, and the experience to know what tool satisfies the problem's requirements and how to apply the proper methods to a problem. Without this help, even if time is not wasted applying a shell to a problem not suitable for an expert system, or applying the shell to a problem not suited for that particular system, time will be spent learning the basic principles of the shell. Time will also be spent learning any other shells purchased to compensate for the shortcomings of the first one chosen.

The ideal solution is an expert system tool which will fit any application. Unfortunately, it is inherent in the nature of a fixed system that it has limitations. One possible alternative is to acquire a shell for which the source code is available. This *open* system could then be altered to fit the requirements of various applications. One such system is CLIPS, a rule-based expert system shell developed by NASA (NASA, 1987). CLIPS uses a forward-chaining inference strategy. As rules become true, they are inserted into a queue. The top-most rule in the queue (the one which has been true the longest) is always the next one fired. The shortcoming in this approach is that, in some cases, major changes might be required in the inference strategy, possibly necessitating a virtual rewrite of the entire inference engine. This might also be the case if another knowledge representation strategy, such as frames, is required. It is probably safe to say that this is a more viable approach than purchasing a smaller closed shell, but unless the open shell fits one's typical requirements closely, this may be as limiting an approach as purchasing a large shell (though considerably cheaper), due to the limitations of its inference strategy and knowledge representation scheme.

Another possible approach is to build the expert system from scratch. DENDRAL (Buchanan, 1969) and MYCIN (Shortliffe, 1976) were built in this manner. The advantage with this approach is that the finished system will be custom fitted to the problem. Additionally, the builders of the system will complete the project with a thorough understanding of expert system design. One problem with this approach is that all of the code written will be designed as an intrinsic part of the system. It might not be feasible to reuse any of it for subsequent applications. Another problem is the time it will take to create the program, since it must be accomplished from scratch. Few expert systems will be built from scratch. The ones that are will be responses to problems not suitable for existing shells, produced without intentions of applying the techniques employed to other problems. Thus these endeavors are likely to be for research purposes only.

A third option is to build one's own shell, making it flexible enough to handle all anticipated applications. This has not been an uncommon approach of late. Several attempts have been made to construct shells which either employ what the designer feels are the "best" inference and knowledge representation methods, or those that are most suited to a particular field (Sorrells, 1985; Takenouchi, 1987). Unfortunately, these systems suffer from the same shortcomings as any other expert system shell: once they are completed, their limitations are built in.

A fourth alternative may be appropriate when large complex problems, for which existing shells are inadequate, are the norm: modular and incremental development. If faced with problems for which expert systems -- or large systems incorporating expert system methods -- must be built from scratch, an attempt should be made to retain as much of each system as possible for subsequent use. This is in essence what has happened with the development of expert system shells. However, commercial shells, as has been described, are static. They offer a particular set of techniques which cannot be altered. It would be desirable if the power of the larger systems could be acquired without their concomitant limitations.

Commercial shell limitations are of two varieties. On one hand, their inference strategy and knowledge representation scheme are fixed. On the other, the data structures and procedural programming techniques that can be used in conjunction with them are limited, thus making it impossible for a programmer to use the conventional methods with which he normally can address any problem.

In order to best retain the elements of an expert system for use in other expert systems without incurring any of these limitations, the elements should be modularized within the system. For example, if a frame-based knowledge representation scheme is used in a given system, but the system's builder wishes to be able to add the capacity for rule-based representation in a subsequent system, he must be aware of how the two will interact and design his inference strategy in an appropriately modular manner. Similarly, he must not code in restrictions on the type of data structures and procedural algorithms that can be interfaced if he wishes to be able to merge his tool with procedural programs. As an example, the part of the inference engine which tests the truth of a rule's antecedent which states "if the identity of an object is x," should be able to refer to a frame hierarchy implemented as a complex data structure, or to one of several other rules, to determine whether the object really is x. The determination of the truth of the antecedent of rules should also allow involved procedural calculations, and the selection of which of the true rules will be fired should be determined by any one of several possible plug-in conflict resolution strategies. These several elements need not be implemented until a problem demands them, but unless their need was anticipated and their incorporation into the system was allowed for in the design of the first-constructed components, it might be difficult to add them. Thus, only the modular components whose need can be foreseen can be designed in from the beginning. This is not as limiting as it sounds. For example, if it is possible to build the system so that it simply sends the numbers of several rules to a conflict resolution strategy as parameters, receiving the number of the rule to fire in return, the builder then has few limitations on the design of conflict-resolution strategies which he might "plug in".

This approach to the problem is, to the author's knowledge, untested. If it can be accomplished, the effort invested would be most rewarded in an environment in which a variety of complex applications will be encountered, each having its own requirements, particularly if the applications require changing or tuning components for efficiency or incorporating the expert systems into multi-system environments. This approach would be most applicable for military applications, in a large industrial setting, or as a tool sold or used by a knowledge engineering firm. It seems a reasonable assumption that the investigation of this approach is justified by the potential returns, not only with respect to the increased flexibility afforded the initial builders of such a tool, but also in regard to the increased expert system building capability that would be disseminated to other users. Further, if researchers elsewhere developed new methods -- a better strategy for abduction, for example -- it might be possible to assimilate the new method without rebuilding a major portion of the system.

8. CONCLUSION

Until expert system technology is as widely understood as traditional computing methods, selecting an expert system shell for a first attempt at building an expert system will be a risky proposition. The degree of risk involved will be influenced by the expense of the shell and the size of the project, and especially by how well or little expert system methods are understood by the ones who choose it. Much of this understanding might be attainable only through experience. For large, complex problems, many shells will be inadequate. This is an even greater concern if it is desirable to apply the tool to multiple problems. In this light, it might be advisable to expend the effort to custom build a tool, with as modular a design as possible. In any event, it is desirable that the fit (or lack of fit) between any closed expert system shell and a potential application be thoroughly understood before the shell is chosen for implementation of the expert system.

ACKNOWLEDGMENTS

I am indebted to L. Wayne Ashby and David A. Brant for the support and guidance they have provided during the course of these projects, and for making the undertaking of this research possible.

I would also like to express my appreciation to Dila Davis for her indispensible and insightful editorial support.

REFERENCES

Al-Zobaide, A., J. B. Grimson (1987). "Expert Systems and Data-Base Systems: How Can They Serve Each Other?" Expert Systems 4(1), 30-37.

Anonymous (1987). "Expert Systems to Triple in '87," Applied Artificial Intelligence Reporter 4(3), 1-8.

Anonymous (1987). "Expert System Interview: Dr. Frederick Hayes-Roth," Expert Systems 4(1), 44-46.

Bobrow, D. G., S. Mittal, M. J. Stefik (1986). "Expert Systems: Perils and Promise," Communications of the ACM 29(9), 880-894.

Brachman, R. J., S. Amarel, C. Engelman, R. A. Engelmore, E. A. Feigenbaum, D. E. Wilkens (1984). "What are Expert Systems?" in Building Expert Systems, F. Hayes-Roth et al. (Addison-Wesley, Reading, Massachusetts), 31-56.

Buchanan, B. G., G. L. Sutherland, E. A. Feigenbaum (1969). "Heuristic DENDRAL: A Program for Generating Explanatory Hypotheses in Organic Chemistry," in Machine Intelligence, Vol. 4, B. Meltzer, D. Michie, eds. (Edinburgh University Press, Edinburgh), 209-254.

Cohen, J. (1985). "Describing Prolog by its Interpretation and Compilation," Communications of the ACM 28(12), 1311-1324.

Cupello, J. M., D. Mishelevich (1988). "Managing Prototype Knowledge/Expert System Projects," Communications of the ACM 31(5), 534-541.

Fikes, R., T. Kehler (1985). The Role of Frame-Based Representation in Reasoning," Communications of the ACM 28(9), 904-920.

Gilmore, J. F. (1985). "Military Applications of Expert Systems," Future Generations Computer Systems 1(6), 403-410.

Hayes-Roth, F. (1984). "The Knowledge-Based Expert System: A Tutorial," Computer 17(9), 11-28.

Kitzmiller, C. T., J. S. Kowalik (1987). "Coupling Symbolic and Numeric Computing in KB Systems," AI Magazine 8(2), 85-90.

Klahr, P. (1980). "Knowledge-Based Simulation," Proceedings of the First Annual National Conference on Artificial Intelligence, Palo Alto, California, 181-183.

Kornell, J. (1987). "Reflections on Using Knowledge Based Systems for Military Simulation," Simulation 48(4), 144-148.

Laffey, T. J., P. D. Cox, J. L. Schmidt, S. M. Kao, J. Y. Read (1988). "Real-Time Knowledge-Based Systems," AI Magazine 9(1), 27-45.

McGracken, D., M. Jackson (1982). "Life Cycle Concept Considered Harmful," Software Engineering Notes 7(2), 31.

Mettrey, W. (1987). "An Assessment of Tools for Building Large Knowledge-Based Systems," AI Magazine 8(4), 81-89.

Michaelson, R. H., D. Michie, A. Boulanger (1985). "The Technology of Expert Systems," Byte Magazine 10(4), 303-314.

Minsky, M. (1975). "A Framework for Representing Knowledge," in The Psychology of Computer Vision, P. Winston, ed. (McGraw-Hill Book Co., Inc., New York).

NASA Johnson Space Center (1987). CLIPS Reference Manual, Artificial Intelligence Section, Houston, Texas.

Newell, A. (1962). "Learning, Generality, and Problem Solving Programs," Rand Memorandum RM-3283-PR, Rand Corp., Santa Monica, California.

Nilsson, N. J. (1980). Principles of Artificial Intelligence (Tioga Press, Palo Alto, California).

Niwa, K., Iham Sasaki (1984). "An Experimental Comparison of Knowledge Representation Schemes," AI Magazine 5(2), 29-37.

Porter, B. W., R. E. Bareiss (1986). "PROTOS: An Experiment in Knowledge Acquisition for Heuristic Classification Tasks," AI Technical Report (AI TR 86-35) Artificial Intelligence Laboratory, The University of Texas at Austin.

Ramsey, C. L., J. A. Reggia, D. S. Nau, A. Ferrentino (1986). "A Comparative Analysis of Methods for Expert Systems," International Journal of Man-Machine Studies 24, 475-499.

Rich, E. (1983). Artificial Intelligence (McGraw-Hill Book Co., Inc., New York).

Robinson, J. A. (1965). "A Machine Oriented Logic Based on the Resolution Principle," Assoc. for Computing Machinery 12(1), 23-41.

Schar, S. (1988). "Intelligent Databases," Database Programming and Design 1(6), 46-53.

Shore, J. (1985). The Sachertorte Algorithm (Viking Press, New York).

Shortliffe, E. H. (1976). Computer-Based Medical Consultation: MYCIN (American Elsevier, New York).

Sorrells, M. E. (1985). "A Time-Constrained Inference Strategy for Real-Time Expert Systems," IEEE National Aeronautics and Electronics Conference, Dayton, Ohio.

Stefik, M., J. Aikens, R. Balzer, J. Benoit, L Birnbaum, F. Hayes-Roth, E. Sacerdot (1983). "Basic Concepts for Building Expert Systems," in Building Expert Systems F. Hayes-Roth et al., eds. (Addison-Wesley, Reading, Massachusetts), 59-86.

Takenouchi, H., Y. Iwashata (1987). "An Integrated Knowledge Representation Schemes," AI Magazine 5(2), 29-37.

Teknowledge (1986). "S.1 Development Reference Manual," Palo Alto, California.

Van De Reit, R. P. (1987). "Problems with Expert Systems?", Future Generations Computer Systems 3, 11-16.

van Melle, W. (1979). "A Domain-Independent Production-Rule System for Consultation Programs," Proceedings of the Sixth International Joint Conference on Artificial Intelligence, 923-925.

Weiss, S. M., C. A. Kulikowski, A. Safir (1978). "A Model-Based Consultation System for the Long Term Management of Glaucoma," Proceedings of the Fifth International Joint Conference on Artificial Intelligence, 826-832.

Weiss, S. M., C. A. Kulikowski (1979). "EXPERT: A System for Developing Consultation Models," Proceedings of the Sixth Annual Conference on Artificial Intelligence, 942-947.

30 May 1989

DISTRIBUTION LIST FOR ARL-TR-89-32 UNDER ARL:UT INDEPENDENT RESEARCH AND DEVELOPMENT PROGRAM

Copy No.	
	Commander
	TEXCOM Field Artillery Board
	1655E Randolf Road
	Fort Sill
	Lawton, OK 73505-6100
1	Attn: N. Bonacci (ATSD)
2	COL Elder (ATZR-BDP)
3	L. Riley (TDSTD)
	Commander
	Department of the Army
	TEXCOM
	Fort Hood, TX 76544-5065
4	Attn: H. Menefee (ATCT-I-D)
5	R. Burn (ATCT-I-D)
	Commander
	TEXCOM Intelligence and Security Board
	Hayes Hall, Fort Huachuca
	Sierra Vista, AZ 85613
6	Attn: W. Woolverton (ATSI-BT-T)
	D. Sabalos (ATSI-BD-AM)
	President
	TEXCOM Air Defense Artillery Board
	Building 1656
	Fort Bliss, TX 79916
7 - 8	Attn: C. Weaver (ATCT-ADA)
	Commanding Officer
	Naval Ocean Research and Development Activity
	Stennis Space Center, MS 39529-5000
9	Attn: Library
	Office of the Chief of Naval Research
	Department of the Navy
	Arlington, VA 22217-5000
10	Attn: M. Orr
11	Library
	Office of the Chief of Naval Research
	Office of Naval Technology
	Department of the Navy
	Arlington, VA 22217-5000
12	Attn: Library

Distribution List for ARL-TR-89-32 under ARL:UT Independent Research and Development Program
(cont'd)

Copy No.	
	Commander
	Space and Naval Warfare Systems Command
	Department of the Navy
	Washington, D.C. 20363-5100
13	Attn: L. Parish (PMW180)
14	K. Evans (PMW180-4)
	Commander
	Naval Sea Systems Command
	Department of the Navy
	Washington, D.C. 20362-5101
15	Attn: E. Plummer (Code 63D1)
	Commander
	Naval Air Systems Command
	Department of the Navy
	Washington, D.C. 20361-5460
16	Attn: Code 933B
	Director
	Naval Research Laboratory
	Washington, D.C. 20375
17	Attn: Library
	Commanding Officer
	Naval Ocean Systems Center
	San Diego, CA 92152-5000
18	Attn: D. Hanna (Code 702)
19	G. Walt (Code 723)
20	Library
	Commander
	Naval Surface Warfare Center
	White Oak Laboratory
	Silver Spring, MD 20910
21	Attn: Library
	Commander
	David Taylor Research Center
	Bethesda, MD 20084
22	Attn: Library
	Commanding Officer
	Naval Oceanographic Office
	Stennis Space Center, MS 39522-5001
23	Attn: Library

Distribution List for ARL-TR-89-32 under ARL:UT Independent Research and Development Program
(cont'd)

Copy No.	
24	Commanding Officer Naval Air Development Center Warminster, PA 18974 Attn: Library
25	Officer in Charge Naval Underwater Systems Center New London Laboratory New London, CT 06320 Attn: Library
26	Superintendent Naval Postgraduate School Monterey, CA 93943 Attn: Library
27	Commanding Officer Naval Coastal Systems Center Panama City, FL 32407 Attn: Library
28	Defense Advanced Research Projects Agency Arlington, VA 22209 Attn: C. Stuart
29	Commanding Officer Naval Intelligence Support Center Washington, D.C. 20390
30 - 41	Commanding Officer and Director Defense Technical Information Center Cameron Station, Building 5 5010 Duke Street Alexandria, VA 22314
42	Applied Research Laboratory The Pennsylvania State University P.O. Box 30 State College, PA 16801 Attn: R. Stern
43	Applied Physics Laboratory The University of Washington 1013 N.E. 40 St. Seattle, WA 98105 Attn: J. Harlett

Distribution List for ARL-TR-89-32 under ARL:UT Independent Research and Development Program
(cont'd)

Copy No.	
44	Marine Physical Laboratory Scripps Institution of Oceanography The University of California, San Diego San Diego, CA 92093 Attn: F. H. Fisher
45	Bell Telephone Laboratories, Inc. Whippany Road Whippany, NJ 07961 Attn: Library
46	Tactical Systems Division, ARL:UT
47	Electromagnetics Group, ARL:UT
48	Garland R. Barnard, ARL:UT
49	David A. Brant, ARL:UT
50	James K. Kroger, ARL:UT
51	Thomas A. Griffy, ARL:UT
52	Library, ARL:UT
53 - 82	Reserve, ARL:UT